How to Use Bitcoin to Play Decentralized Poker

Iddo Bentov
Technion

Ranjit Kumaresan
MIT

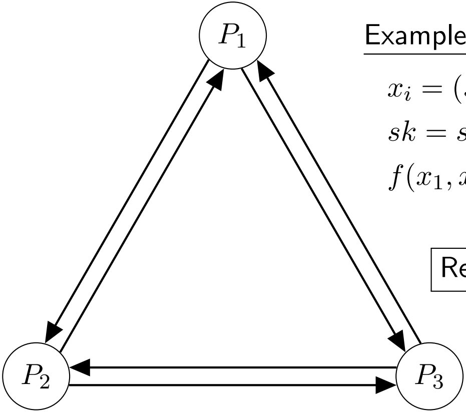
Tal Moran
IDC

GTACS

January 8, 2015

Secure multiparty computation (MPC) / secure function evaluation (SFE)

Parties $P_1, P_2, \ldots, P_n$ with inputs $x_1, x_2, \ldots, x_n$ send messages to each other, and wish to ***securely*** compute $f(x_1, x_2, \ldots, x_n)$.



Example of SFE:

$$x_i = (sk_i, c)$$

$$sk = sk_1 \oplus sk_2 \oplus \cdots \oplus sk_n$$

$$f(x_1, x_2, \ldots, x_n) = \texttt{decrypt}(sk, c)$$

Reactive MPC: think of poker cards

Impossibility of fair MPC

Fairness: if any party receives the output, then all honest parties must receive the output.

"Security with abort" is possible

- Secure MPC is possible [Yao86, GMW87, ...]
  - Security: correctness, privacy, independence of inputs, ~~fairness~~
  - Even with dishonest majority, in the computational setting.

Full security is impossible

- Fair MPC is impossible [Cle86]
  - r-round 2-party coin toss protocol is susceptible to $\Omega(1/r)$ bias.
  - $\Rightarrow$ no fair protocol for XOR, barring gradual release [...]

Our results

Outline of this presentation

1. Impose fairness for any SFE, without an honest majority.
2. Secure (reactive) MPC with money inputs and outputs.
   - Example: poker.

Formal model that incorporates coins

Functionality $\mathcal{F}_\square$ versus functionality $\mathcal{F}_\square^\star$ with coins

- If party P_i has (say) secret key sk_0 and sends it to party P_j, then both P_i and P_j will have the string sk_0.
- If party P_i has coins(x) and sends $y < x$ coins to party P_j, then P_i will have coins$(x - y)$ and P_j will have extra coins(y).

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.

Formal model that incorporates coins

Functionality $\mathcal{F}_\square$ versus functionality $\mathcal{F}^\star_\square$ with coins

- If party P_i has (say) secret key sk_0 and sends it to party P_j, then both P_i and P_j will have the string sk_0.
- If party P_i has coins(x) and sends $y < x$ coins to party P_j, then P_i will have coins$(x - y)$ and P_j will have extra coins(y).

- With Bitcoin: the parties only send strings, but miners do PoW so that the coin transfers become irreversible.
- Ideally, all the parties deem coins to be valuable assets.
- Sending coins(x) may require a broadcast that reveals at least the amount x (maybe not in ZK cryptocurrency like Zerocash).
- It is possible to define a "secure computation with coins" model directly, or by using (UC) ideal functionalities.
- We provide simulation based proofs (but not in this talk).

Claim-or-Refund for two parties P_s,P_r (implicit in [Max11],[BBSU12])

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ Claim-or-Refund ideal functionality

1. The sender P_s deposits (locks) her coins(q) while specifying a time bound τ and a circuit $\phi(\cdot)$.
2. The receiver P_r can claim (gain possession) of the coins(q) by publicly revealing a witness w that satisfies $\phi(w) = 1$.
3. If P_r didn't claim within time τ, coins(q) are refunded to P_s.

Claim-or-Refund for two parties P_s,P_r (implicit in [Max11],[BBSU12])

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ Claim-or-Refund ideal functionality

1. The sender P_s deposits (locks) her coins(q) while specifying a time bound τ and a circuit $\phi(\cdot)$.
2. The receiver P_r can claim (gain possession) of the coins(q) by publicly revealing a witness w that satisfies $\phi(w) = 1$.
3. If P_r didn't claim within time τ, coins(q) are refunded to P_s.

How to realize $\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin

- Old version: using "timelock" transactions.
- New version: `OP_CHECKLOCKTIMEVERIFY` (abbrv. `CLTV`) enables $\mathcal{F}^{\star}_{\mathrm{CR}}$ directly, avoiding transaction malleability attacks.

$\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin (without `CLTV`)

High-level description the $\mathcal{F}^{\star}_{\mathrm{CR}}$ implementation in Bitcoin

- P_s controls TX_{old} that resides on the blockchain.
- P_s creates a transaction TX_{new} that spends TX_{old} to a Bitcoin script that can be redeemed by P_s and P_r, or only by P_r by supplying a witness w that satisfies $\phi(w) = 1$.
- P_s asks P_r to sign a timelock transaction that refunds TX_{new} to P_s at time τ (conditioned upon both P_s and P_r signing).
- After P_r signs the refund, P_s can safely broadcast TX_{new}.

1. P_s is safe because P_r only sees Hash(TX_{new}), and therefore cannot broadcast TX_{new} to cause P_s to lose the coins.
2. P_r can safely sign the random-looking data Hash(TX_{new}) because the protocol uses a freshly generated (sk_R, pk_R) pair.

The structure of Bitcoin transactions

How standard Bitcoin transactions are chained

- TX_{old} = earlier TX output of coins(q) is redeemable by pk_A
- $id_{\text{old}} = \text{Hash}(TX_{\text{old}})$
- $PREPARE_{\text{new}} = (id_{\text{old}}, q, pk_B, 0)$ 0 means no timelock
- $TX_{\text{new}} = (PREPARE_{\text{new}}, \ \texttt{Sign}_{sk_A}(PREPARE_{\text{new}}))$
- $id_{\text{new}} = \text{Hash}(TX_{\text{new}})$
- Initial minting transaction specifies some pk_M that belongs to a miner, and is created via *proof of work*.

Realization of $\mathcal{F}^{\star}_{\mathrm{CR}}$ via Bitcoin (without CLTV)

The $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction

- $PREPARE_{\textsf{new}} = (id_{\textsf{old}}, q, (pk_S \wedge pk_R) \vee (\phi(\cdot) \wedge pk_R), 0)$
- $\phi(\cdot)$ can be $\texttt{SHA256}(\cdot) == Y$ where Y is hardcoded.
- $TX_{\textsf{new}} = (PREPARE_{\textsf{new}},\ \texttt{Sign}_{sk_S}(PREPARE_{\textsf{new}}))$
- $id_{\textsf{new}} = \textsf{Hash}(TX_{\textsf{new}})$
- P_s sends $PREPARE_{\textsf{refund}} = (id_{\textsf{new}}, q, pk_S, \tau)$ to P_r
- P_r sends $\sigma_R = \texttt{Sign}_{sk_R}(PREPARE_{\textsf{refund}})$ to P_s
- P_s broadcasts $TX_{\textsf{new}}$ to the Bitcoin network
- If P_r doesn't reveal w until time τ then P_s creates $TX_{\textsf{refund}} = (PREPARE_{\textsf{refund}},\ (\texttt{Sign}_{sk_S}(PREPARE_{\textsf{refund}}), \sigma_R))$ and broadcasts it to reclaim her q coins

$\mathcal{F}^{\star}_{\text{CR}}$ via Bitcoin with CLTV (operational since $\approx$ December 2015)

Pseudocode: pk_S, pk_R, h_0, τ are hardcoded

`if (block# >` τ`) then`
  P_s `can spend the` coins(q) `by signing with` sk_s
`else`
  P_r `can spend the` coins(q) `by`
    `signing with` sk_r
    `AND`
    `supplying` w `such that` Hash$(w) = h_0$ ← `this is` $\phi(\cdot)$

Bitcoin script

```
IF <timeout> CHECKLOCKTIMEVERIFY
   HASH256 <h_0> EQUALVERIFY <pk_r> CHECKSIGVERIFY
ELSE
   <pk_s> CHECKSIGVERIFY
ENDIF
```

Fairness with penalties

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn't deliver output to honest parties $\Rightarrow$ every honest party is compensated

Fairness with penalties

Definition of fair secure multiparty computation with penalties

- An honest party never has to pay any penalty
- If a party aborts after learning the output and doesn't deliver output to honest parties $\Rightarrow$ every honest party is compensated

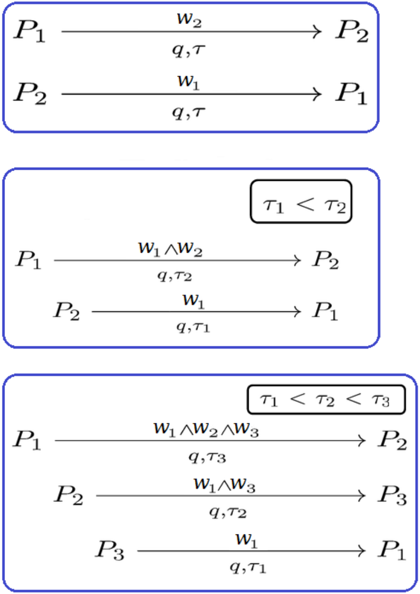
Outline of $\mathcal{F}_f^\star$ – fairness with penalties for any function f

- $P_1, \ldots, P_n$ with $x_1, \ldots, x_n$ run secure *unfair* SFE for f that
  1. Computes additive shares $(y_1, \ldots, y_n)$ of $y = f(x_1, \ldots, x_n)$
  2. Computes $\mathrm{Tags} = (\mathsf{com}(y_1), \ldots, \mathsf{com}(y_n))$ $= (\mathtt{hash}(y_1), \ldots, \mathtt{hash}(y_n))$
  3. Delivers (y_i, Tags) to every P_i
- $P_1, \ldots, P_n$ deposit coins and run fair reconstruction (fair exchange) with penalties to swap the y_i's among themselves.

Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction

"Abort" attack:

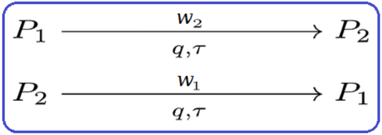
P_2 claims without deposting



Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction

"Abort" attack:

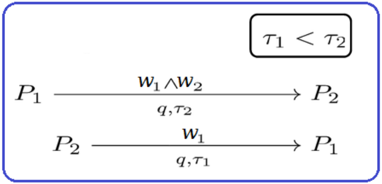
P_2 claims without deposting

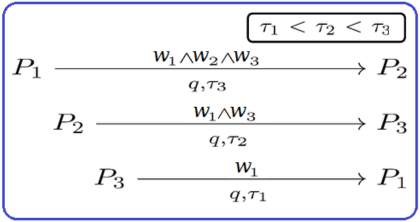


Fair exchange:

P_1 claims by revealing w_1

$\Rightarrow$ P_2 can claim by revealing w_2

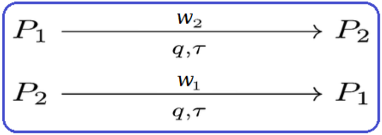




Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction

"Abort" attack:

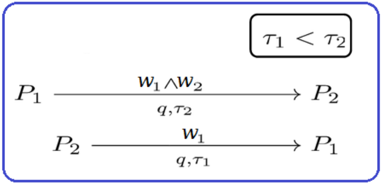
P_2 claims without deposting



Fair exchange:

P_1 claims by revealing w_1

$\Rightarrow$ P_2 can claim by revealing w_2

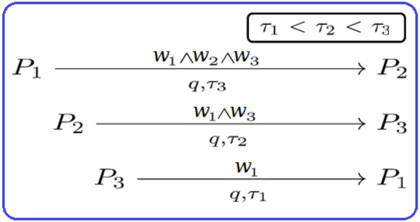


Malicious coalition:

Coalition P_1, P_2 obtain w_3 from P_3

P_2 doesn't claim the top transaction

P_3 isn't compensated



Fair exchange in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model - the ladder construction (contd.)

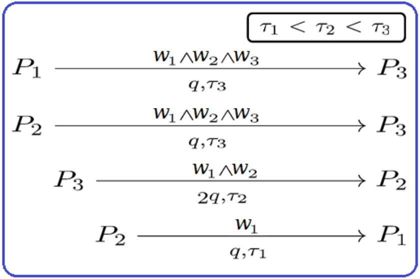
Fair exchange:

Bottom two levels:

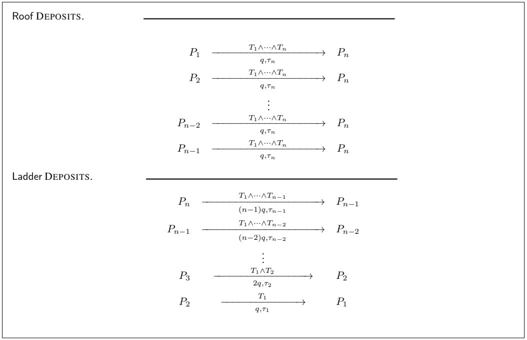
P_1, P_2 get compensated by P_3

Top two levels:

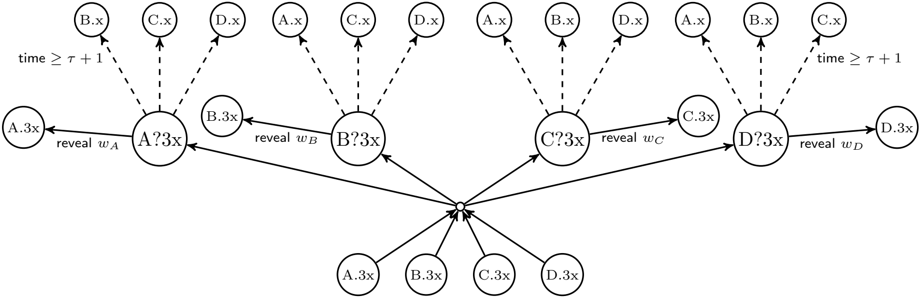
P_3 gets her refunds by revealing w_3



Full ladder:



Multilock



In principle, jointly locking coins for fair exchange can work well:

1. $M =$ "if P_1, P_2, P_3, P_4 sign this message with inputs of coins$(3x)$ each then their $3x$ coins are locked into 4 outputs of coins$(3x)$ each, where each P_i can redeem output T_i with a witness w_i that satisfies ϕ_i, and after time τ anyone can divide an unredeemed output T_i equally to $\{P_1, P_2, P_3, P_4\} \setminus \{P_i\}$"
2. P_1, P_2, P_3, P_4 sign M and broadcast it, and after M is confirmed, each P_i redeems coins(x) by revealing w_i

Practicality of multiparty fair exchange with penalties in Bitcoin

- Unfortunately, $\mathcal{F}^{\star}_{\mathrm{ML}}$ cannot be implemented in vanilla Bitcoin because of self-imposed “transaction malleability” (ECDSA is a randomized signature algorithm).
- Instead, we propose a protocol enhancement that eliminates transaction malleability while retaining expressibility.

Practicality of multiparty fair exchange with penalties in Bitcoin

- Unfortunately, $\mathcal{F}^{\star}_{\mathrm{ML}}$ cannot be implemented in vanilla Bitcoin because of self-imposed "transaction malleability" (ECDSA is a randomized signature algorithm).
- Instead, we propose a protocol enhancement that eliminates transaction malleability while retaining expressibility.

Recap:

- $\mathcal{F}^{\star}_{\mathrm{ML}}$ requires $O(1)$ Bitcoin rounds and $O(n^2)$ transaction data (and $O(n^2)$ signature operations), while the ladder requires $O(n)$ Bitcoin rounds and $O(n)$ transactions.
- Multiparty fair computation can be implemented in Bitcoin via the ladder construction.
- Multiparty fair computation can be implemented via $\mathcal{F}^{\star}_{\mathrm{ML}}$ with an enhanced Bitcoin protocol.

Comparison with other ways to achieve fairness

Gradual release

- Release the output bit by bit...
- Even with only 2 parties, the number of rounds depends on a security parameter.
- Complexity blowup because the protocol must ensure that the parties don't release junk bits.
- Assumptions on the computational power of the parties, sequential puzzles to avoid parallelization.

Comparison with other ways to achieve fairness

Gradual release

- Release the output bit by bit...
- Even with only 2 parties, the number of rounds depends on a security parameter.
- Complexity blowup because the protocol must ensure that the parties don't release junk bits.
- Assumptions on the computational power of the parties, sequential puzzles to avoid parallelization.

Trusted bank

- Legally Enforceable Fairness [Lindell 2008]
- Requires a trusted party to provide an ideal bank functionality.
- 2-party only: the bank can provide $\mathcal{F}^{\star}_{\mathrm{CR}}$ or $\mathcal{F}^{\star}_{\mathrm{ML}}$ to use our constructions directly, or implement similar protocols.
- Not a secure cash distribution protocol...

Secure cash distribution and poker

How to Use Bitcoin to Play Decentralized Poker

Iddo Bentov
Technion

Ranjit Kumaresan
MIT

Tal Moran
IDC

CCS 2015

"Paradoxical" Abilities 1983-

- Exchanging Secret Messages without Ever Meeting
- Simultaneous Contract Signing Over the Phone
- Generating exponentially long pseudo random strings indistinguishable from random
- Proving a theorem without revealing the proof
- $\Longrightarrow$ Playing any digital game without referees
- Private Information Retrieval

Secure cash distribution with penalties

Ideal 2-party secure (non-reactive) cash distribution functionality:

1. Wait to receive $(x_1, \text{coins}(d_1))$ from P_1 and $(x_2, \text{coins}(d_2))$ from P_2.
2. Compute $(y, v) \leftarrow f(x_1, x_2, d_1, d_2)$.
3. Send $(y, \text{coins}(v))$ to P_1 and $(y, \text{coins}(d_1 + d_2 - v))$ to P_2.

Secure cash distribution with penalties

Ideal 2-party secure (non-reactive) cash distribution functionality:

1. Wait to receive $(x_1, \text{coins}(d_1))$ from P_1 and $(x_2, \text{coins}(d_2))$ from P_2.
2. Compute $(y, v) \leftarrow f(x_1, x_2, d_1, d_2)$.
3. Send $(y, \text{coins}(v))$ to P_1 and $(y, \text{coins}(d_1 + d_2 - v))$ to P_2.

- In the general case, each party P_i has input $(x_i, \text{coins}(d_i))$ and receives output $(y, \text{coins}(v_i))$.
- Use-cases: generalized lottery, incentivized computation, . . .

Blackbox secure cash distribution

- Blackbox realization of secure cash distribution in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model.
- Assume: the input coins amount of P_i is an m_i-bit number.

Step 1: commit to random secrets (preprocessing)

For all $i \in [n], j \in [n] \setminus \{i\}, k \in [m_i]$:

- P_i picks a random witness $w_{i,j,k} \leftarrow \{0,1\}^\lambda$
- P_i computes $c_{i,j,k} \leftarrow \mathsf{commit}(1^\lambda, w_{i,j,k})$.
- P_i sends $c_{i,j,k}$ to all parties.
- P_i makes an $\mathcal{F}^{\star}_{\mathrm{CR}}$ transaction $P_i \xrightarrow[2^k,\tau]{w_{i,j,k}} P_j$

Blackbox secure cash distribution (contd.)

Denote the the input coin amounts by $d = (d_1, \ldots, d_n)$ and the string inputs by $(x_1, x_2, \ldots, x_n)$.

Step 2: compute the cash distribution

Invoke secure SFE (unfair for now) for the cash distribution:

- Compute the output coin amounts $v = (v_1, v_2, \ldots, v_n)$.
- Derive numbers $b_{i,j}$ that specify how many coins P_i needs to send P_j according to the input coins d and output coins v.
- Let $(b_{i,j,1}, b_{i,j,2}, \ldots, b_{i,j,m_i})$ be the binary expansion of $b_{i,j}$.
- For all i, j, k, if $b_{i,j,k} = 1$ then concatenate to the output a value $w'_{i,j,k}$ that satisfies commit$(1^\lambda, w'_{i,j,k}) = c_{i,j,k}$.
- Compute $y = f(x_1, x_2, \ldots, x_n)$ and output y too.

Then, use fair exchange with penalties (with time limit $< \tau$) to deliver the output to all parties, so that $\mathcal{F}^\star_{\mathrm{CR}}$ claims will ensue.

Is one-shot protocol enough?

Are we there yet?

Is one-shot protocol enough?

Are we there yet? In the case of poker, not really.

- The most natural formulation of poker is as a *reactive* secure MPC.
- Multistage protocol: after each stage of the computation some intermediate outputs are revealed to the parties.
  - Example: the top card of the deck is revealed to all parties.
- One-shot protocol is not the natural formulation:
  - A circuit that takes into account all the possible variables is highly inefficient.
  - Those variables may depend on external events (say, you receive a phone call regarding an unrelated financial loss).
- $\Rightarrow$ must be dropout-tolerant:
  - After a stage that reveals information, corrupt parties must be penalized if they abort.
  - In fact, the corrupt parties must be penalized unless they continue the next stage of the computation.

Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.

Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.
- The given secure MPC (whitebox) where for every round r a single message is broadcast by a designated party P_{i_r}.

Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.
- The given secure MPC (whitebox) where for every round r a single message is broadcast by a designated party P_{i_r}.
- $\mathcal{F}^{\star}_{\mathrm{CR}}$ transactions $P_i \xrightarrow[q,\tau]{\phi_{i,j}} P_j$ where $\phi_{i,j}$ is a circuit (script) that is satisfied if P_i created multiple signed extensions of protocol's execution (with a unique starting nonce).

Reactive secure cash distribution

Ingredients needed:

- See-saw instead of the ladder construction, to force parties to make the next move.
- The given secure MPC (whitebox) where for every round r a single message is broadcast by a designated party P_{i_r}.
- $\mathcal{F}^{\star}_{\mathrm{CR}}$ transactions $P_i \xrightarrow[q,\tau]{\phi_{i,j}} P_j$ where $\phi_{i,j}$ is a circuit (script) that is satisfied if P_i created multiple signed extensions of protocol's execution (with a unique starting nonce).
- Blackbox secure cash distribution as described, with refunds at time τ that exceeds the see-saw time limits, and hence with circuits specified at start that are utilized in the final rounds.

The see-saw construction: 2 parties

ROOF DEPOSIT.

$$P_1 \xrightarrow[q,\tau_{m,2}]{\mathrm{TT}_{m,2}} P_2 \qquad (\mathsf{Tx}_{m,2})$$

SEE-SAW DEPOSITS. For $r = m - 1$ to 1:

$$P_2 \xrightarrow[2q,\tau_{r+1,1}]{\mathrm{TT}_{r+1,1}} P_1 \qquad (\mathsf{Tx}_{r+1,1})$$

$$P_1 \xrightarrow[2q,\tau_{r,2}]{\mathrm{TT}_{r,2}} P_2 \qquad (\mathsf{Tx}_{r,2})$$

FLOOR DEPOSIT.

$$P_2 \xrightarrow[q,\tau_{1,1}]{\mathrm{TT}_{1,1}} P_1 \qquad (\mathsf{Tx}_{1,1})$$

The see-saw construction: multiparty

Roof deposits. For each $j \in [n-1]$:

$$P_j \xrightarrow[q,\tau_{2n-2}]{\mathrm{TT}_n} P_n$$

Ladder deposits. For $i = n-1$ down to 2:

- Rung unlock: For $j = n$ down to $i+1$:

$$P_j \xrightarrow[q,\tau_{2i-1}]{\mathrm{TT}_i \wedge U_{i,j}} P_i$$

- Rung climb:

$$P_{i+1} \xrightarrow[i \cdot q,\tau_{2i-2}]{\mathrm{TT}_i} P_i$$

- Rung lock: For each $j = n$ down to $i+1$:

$$P_i \xrightarrow[q,\tau_{2i-2}]{\mathrm{TT}_{i-1} \wedge U_{i,j}} P_j$$

Foot deposit.

$$P_2 \xrightarrow[q,\tau_1]{\mathrm{TT}_1} P_1$$

The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With m rounds, $O(n^2m)$ calls to $\mathcal{F}^{\star}_{\mathrm{CR}}$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.

The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With m rounds, $O(n^2m)$ calls to $\mathcal{F}^{\star}_{\mathrm{CR}}$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
- Party P_i who aborts pays compensation to all other parties.
- In the ladder P_i can abort and then nobody learns the secret.

The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With m rounds, $O(n^2m)$ calls to $\mathcal{F}^{\star}_{\text{CR}}$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
- Party P_i who aborts pays compensation to all other parties.
- In the ladder P_i can abort and then nobody learns the secret.
- This is crucial for reactive functionalities:
  - Consider poker: suppose that in round j all parties exchange shares to reveal the top card of the deck.
  - If P_i didn't like this top card, we must not allow P_i to abort in round $j+1$ without punishment.

The see-saw construction: multiparty (contd.)

Properties of the multiparty see-saw

- With m rounds, $O(n^2m)$ calls to $\mathcal{F}^{\star}_{\mathrm{CR}}$ (ladder is $O(nm)$).
- $O(nm)$ security deposit by each party.
- Party P_i who aborts pays compensation to all other parties.
- In the ladder P_i can abort and then nobody learns the secret.
- This is crucial for reactive functionalities:
  - Consider poker: suppose that in round j all parties exchange shares to reveal the top card of the deck.
  - If P_i didn't like this top card, we must not allow P_i to abort in round $j+1$ without punishment.
- The circuits verify a signed extension of the entire execution transcript, and that this extension conforms with the protocol.
- $\Rightarrow$ needs more expressive scripting language than vanilla Bitcoin, but not Turing complete scripts because the round bounds are known in advance.

The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.

The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.
- Preprocessing step: make the cash distribution transactions with random circuits $w_{i,j,k}$.
- Invoke (preprocess) at start an unfair SFE that:
  - Shuffles the deck according to the parties' random inputs.
  - Computes commitments to shares of all the cards.
  - Deals shares of the hands and shares of the rest of the cards to all parties, and also delivers all the commitments to all parties.

The see-saw construction: poker

- No need to run reactive secure MPC that corresponds to rounds of the see-saw.
- Preprocessing step: make the cash distribution transactions with random circuits $w_{i,j,k}$.
- Invoke (preprocess) at start an unfair SFE that:
  - Shuffles the deck according to the parties' random inputs.
  - Computes commitments to shares of all the cards.
  - Deals shares of the hands and shares of the rest of the cards to all parties, and also delivers all the commitments to all parties.
- The $\mathcal{F}^{\star}_{\mathrm{CR}}$ circuit in each round of the see-saw will verify signatures of a transcript, then enforce betting rules or force a party to reveal a share of a card, or in the final round force a party to reveal some $w_{i,j,k}$ values.
- For example: if all partied called and the top card on the deck should be revealed, then the next see-saw circuits will require each party to reveal her share of the top card.

Some open questions

- Lower bound of linear number of rounds for fairness with penalties in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model?
- Constructing secure cash distribution with penalties from *blackbox* secure MPC and $\mathcal{F}^{\star}_{\mathrm{CR}}$?

Some open questions

- Lower bound of linear number of rounds for fairness with penalties in the $\mathcal{F}^{\star}_{\mathrm{CR}}$-hybrid model?
- Constructing secure cash distribution with penalties from *blackbox* secure MPC and $\mathcal{F}^{\star}_{\mathrm{CR}}$?

Thank you.

version 4